Exhibit 7

Edition

RNETWORKING WITH

CP/IP VOLUME I PRINCIPLES, PROTOCOLS, AND ARCHITECTURE

DOUGLAS E. COMER

0,000 Copies Sold

assic text for an introduction to TCP/IP."
—Jon Postel, RFC editor and former Deputy Internet Architect

gh others have tried, there is no better written or organized explanation of the core
P."
—Joel Snyder, *Network Computing*

ntroduction to the TCP/IP protocol suite and its underpinnings, this is an excellent
is also a good reference book to keep around for anyone who is working with
—George V. Neville-Neil, *USENIX ;login:*

ime best-selling TCP/IP book, *Internetworking with TCP/IP*, is still THE reference
ne who wants to learn about or work with the TCP/IP protocol suite. Volume I of
s by Douglas Comer provides the most up-to-date conceptual introduction to
protocols and the latest developments in Internet technology.

ed for its clarity and accessibility, this superb text covers wide area (WAN) Internet
nes as well as local area network (LAN) technologies like Ethernet and FDDI. The
lains address binding (ARP), IP connectionless datagram delivery, error detection,
ting, and routing.

EW EDITION OF VOLUME I:

sses how to use TCP/IP over an ATM network.
s the latest IPng (next generation) developments and information.
ibes CIDR (Classless Inter-Domain Routing) and supernetting.
sses security in TCP/IP environments and firewall design.
orizes hundreds of new RFCs and the protocols they describe.

ion, Volume I:

ares the ISO 7-layer reference model to the TCP/IP 5-layer reference model.
ns TCP: reliability, acknowledgments, flow control, and sliding windows.
s adaptive retransmission, including slow-start and silly window avoidance.
ibes the socket interface that applications use to access TCP/IP protocols.
nts routing architectures for large and small internets.
sses bridges and routers.
nes application services:
main Name System (DNS)
ctronic mail (SMTP, MIME)
transfer and access (FTP, TFTP, NFS)
note login (TELENET, rlogin)
work management (SNMP, MIB, ANS.1)

CE HALL

ISBN 0-13-216987-8
90000

COMER

Third Edition

INTERNETWORKING WITH TCP/IP VOLUME I PRINCIPLES, PROTOCOLS, AND ARCHITECTURE

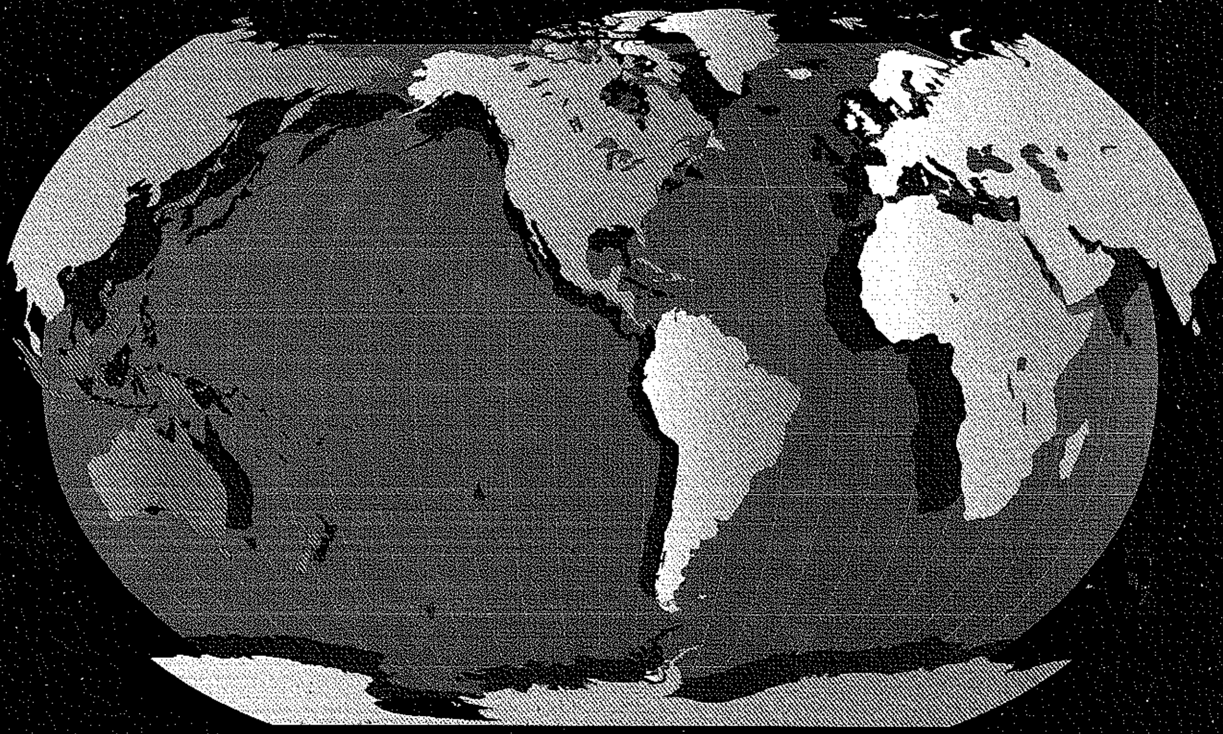
Third Edition

INTERNETWORKING WITH TCP/IP

Includes IPng and IP over ATM

VOLUME I

PRINCIPLES, PROTOCOLS, AND ARCHITECTURE



Internetworking With TCP/IP

Vol I:

Principles, Protocols, and Architecture

Third Edition

DOUGLAS E. COMER

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

PRENTICE HALL

Library of Congress Cataloging-in-Publication Data

Comer, Douglas
Internetworking with TCP/IP / Douglas E. Comer. -- 3rd ed.
p. cm.
Includes bibliographical references and index.
Contents: v. 1. Principles, protocols, and architecture
ISBN 0-13-216987-8 (v. 1)
1. TCP/IP (Computer network protocol) 2. Client/server computing.
3. Internetworking (Telecommunication) I. Title.
TK5105.585.C66 1995
005.2--dc20 95-1830
CIP

Acquisitions editor: ALAN APT
Production editor: IRWIN ZUCKER
Cover designer: WENDY ALLING JUDY
Buyer: LORI BULWIN
Editorial assistant: SHIRLEY MCGUIRE

Printed in the United States of America

10 9 8 7 6 5

ISBN 0-13-216987-8

Prentice-Hall International (UK) Limited, London
Prentice-Hall of Australia Pty. Limited, Sydney
Prentice-Hall Canada Inc., Toronto
Prentice-Hall Hispanoamericana, S.A., Mexico
Prentice-Hall of India Private Limited, New Delhi
Prentice-Hall of Japan, Inc., Tokyo
Simon & Schuster Asia Pte. Ltd., Singapore

To Chris

11

Protocol Layering

11.1 Introduction

Previous chapters review the architectural foundations of internetworking, describe how hosts and routers forward Internet datagrams, and present mechanisms used to map IP addresses to physical network addresses. This chapter considers the structure of the software found in hosts and routers that carries out network communication. It presents the general principle of layering, shows how layering makes Internet Protocol software easier to understand and build, and traces the path of datagrams through the protocol software they encounter when traversing a TCP/IP internet.

11.2 The Need For Multiple Protocols

We have said that protocols allow one to specify or understand communication without knowing the details of a particular vendor's network hardware. They are to computer communication what programming languages are to computation. It should be apparent by now how closely the analogy fits. Like assembly language, some protocols describe communication across a physical network. For example, the details of the Ethernet frame format, network access policy, and frame error handling comprise a protocol that describes communication on an Ethernet. Similarly, the details of IP addresses, the datagram format, and the concept of unreliable, connectionless delivery comprise the Internet Protocol.

Complex data communication systems do not use a single protocol to handle all transmission tasks. Instead, they require a set of cooperative protocols, sometimes called a *protocol family* or *protocol suite*. To understand why, think of the problems that arise when machines communicate over a data network:

• *Hardware failure.* A host or router may fail either because the hardware fails or because the operating system crashes. A network transmission link may fail or accidentally be disconnected. The protocol software needs to detect such failures and recover from them if possible.

• *Network congestion.* Even when all hardware and software operates correctly, networks have finite capacity that can be exceeded. The protocol software needs to arrange ways that a congested machine can suppress further traffic.

• *Packet delay or loss.* Sometimes, packets experience extremely long delays or are lost. The protocol software needs to learn about failures or adapt to long delays.

• *Data corruption.* Electrical or magnetic interference or hardware failures can cause transmission errors that corrupt the contents of transmitted data. Protocol software needs to detect and recover from such errors.

• *Data duplication or sequence errors.* Networks that offer multiple routes may deliver data out of sequence or may deliver duplicates of packets. The protocol software needs to reorder packets and remove any duplicates.

Taken together, all these problems seem overwhelming. It is difficult to understand how to write a single protocol that will handle them all. From the analogy with programming languages, we can see how to conquer the complexity. Program translation has been partitioned into four conceptual subproblems identified with the software that handles each subproblem: compiler, assembler, link editor, and loader. The division makes it possible for the designer to concentrate on one subproblem at a time, and for the implementor to build and test each piece of software independently. We will see that protocol software is partitioned similarly.

Two final observations about our programming language analogy will help clarify the organization of protocols. First, it should be clear that pieces of translation software must agree on the exact format of data passed between them. For example, the data passed from the compiler to the assembler consists of a program defined by the assembly programming language. Thus, we see how the translation process involves multiple programming languages. The analogy will hold for communication software, where we will see that multiple protocols define the interfaces between the modules of communication software. Second, the four parts of the translator form a linear sequence in which output from the compiler becomes input to the assembler, and so on. Protocol software also uses a linear sequence.

11.3 The Conceptual Layers Of Protocol Software

Think of the modules of protocol software on each machine as being stacked vertically into *layers*, as in Figure 11.1. Each layer takes responsibility for handling one part of the problem.

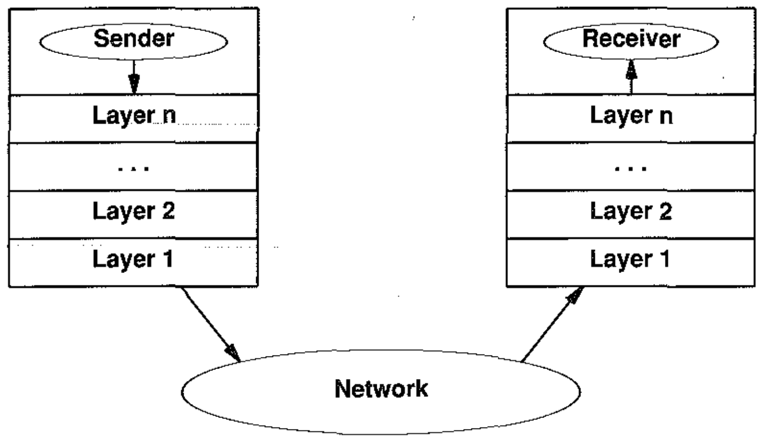


Figure 11.1 The conceptual organization of protocol software in layers.

Conceptually, sending a message from an application program on one machine to an application program on another means transferring the message down through successive layers of protocol software on the sender's machine, transferring the message across the network, and transferring the message up through successive layers of protocol software on the receiver's machine.

In practice, the protocol software is much more complex than the simple model of Figure 11.1 indicates. Each layer makes decisions about the correctness of the message and chooses an appropriate action based on the message type or destination address. For example, one layer on the receiving machine must decide whether to keep the message or forward it to another machine. Another layer must decide which application program should receive the message.

To understand the difference between the conceptual organization of protocol software and the implementation details, consider the comparison shown in Figure 11.2. The conceptual diagram in Figure 11.2a shows an Internet layer between a high level protocol layer and a network interface layer. The realistic diagram in Figure 11.2b shows that the IP software may communicate with multiple high-level protocol modules and with multiple network interfaces.

Although a diagram of conceptual protocol layering does not show all details, it does help explain the general ideas. For example, Figure 11.3 shows the layers of protocol software used by a message that traverses three networks. The diagram shows only the network interface and Internet Protocol layers in routers because only those layers are needed to receive, route, and then send datagrams. We understand that any machine attached to two networks must have two network interface modules, even though the conceptual layering diagram shows only a single network interface layer in each machine.

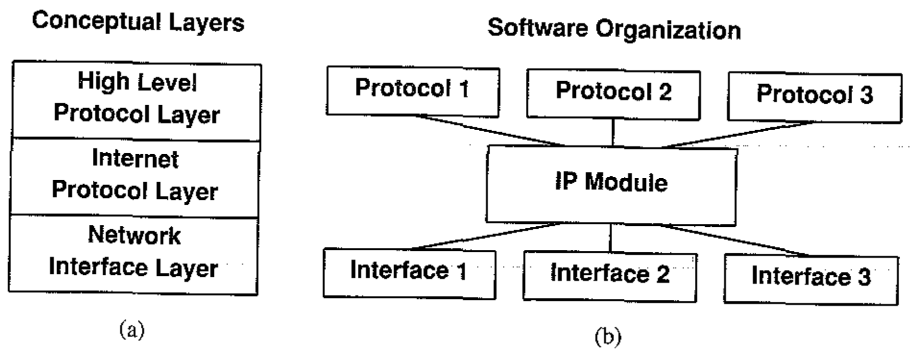


Figure 11.2 A comparison of (a) conceptual protocol layering and (b) a realistic view of software organization showing multiple network interfaces below IP and multiple protocols above it.

As Figure 11.3 shows, a sender on the original machine transmits a message which the IP layer places in a datagram and sends across network *1*. On intermediate machines the datagram passes up to the IP layer which routes it back out again (on a different network). Only when it reaches the final destination machine does IP extract the message and pass it up to higher layers of protocol software.

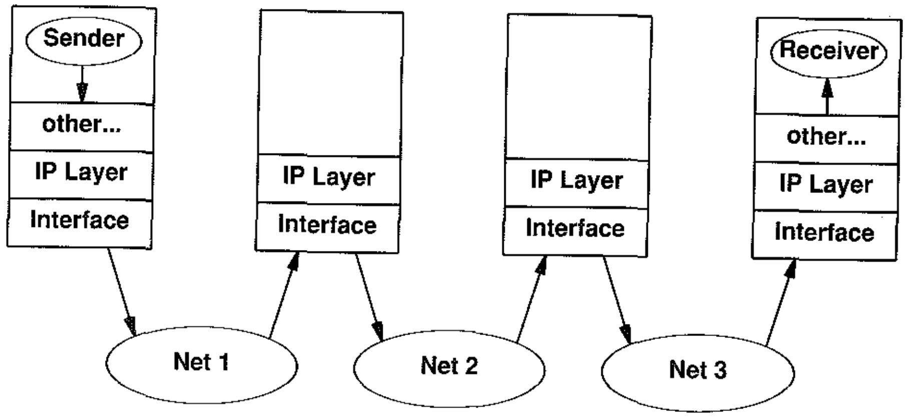


Figure 11.3 The path of a message traversing the Internet from the sender through two intermediate machines to the receiver. Intermediate machines only send the datagram to the IP software layer.

11.4 Functionality Of The Layers

Once the decision has been made to partition the communication problem into subproblems and organize the protocol software into modules that each handle one subproblem, the question arises: ''what functionality should reside in each module?'' The question is not easy to answer for several reasons. First, given a set of goals and constraints governing a particular communication problem, it is possible to choose an organization that will optimize protocol software for that problem. Second, even when considering general network-level services such as reliable transport, it is possible to choose from among fundamentally distinct approaches to solving the problem. Third, the design of network (or internet) architecture and the organization of the protocol software are interrelated; one cannot be designed without the other.

11.4.1 ISO 7-Layer Reference Model

Two ideas about protocol layering dominate the field. The first, based on work done by the International Organization for Standardization (ISO), is known as ISO's *Reference Model of Open System Interconnection*, often referred to as the *ISO model*. The ISO model contains 7 conceptual layers organized as Figure 11.4 shows.

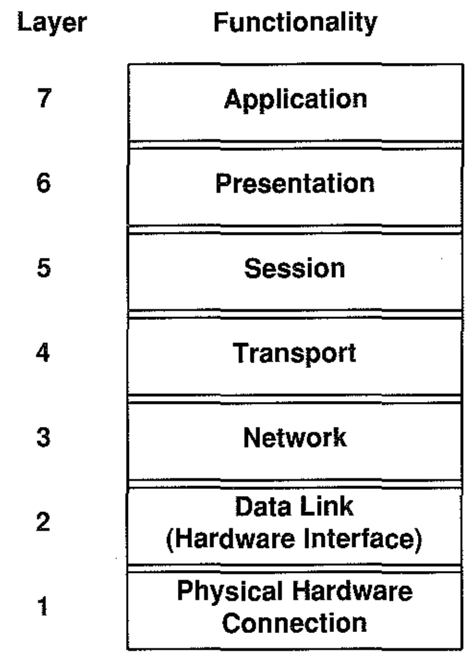


Figure 11.4 The ISO 7-layer reference model for protocol software.

The ISO model, built to describe protocols for a single network, does not contain a specific level for internetwork routing in the same way TCP/IP protocols do.

11.5 X.25 And Its Relation To The ISO Model

Although it was designed to provide a conceptual model and not an implementation guide, the ISO layering scheme has been the basis for several protocol implementations. Among the protocols commonly associated with the ISO model, the set of protocols known as X.25 is probably the best known and most widely used. X.25 was established as a recommendation of the *Telecommunications Section* of the *International Telecommunications Union*† (ITU-TS), an international organization that recommends standards for international telephone services. X.25 has been adopted by public data networks, and is especially popular in Europe. Considering X.25 will help explain the ISO layering.

In the X.25 view, a network operates much like a telephone system. An X.25 network is assumed to consist of complex packet switches that contain the intelligence needed to route packets. Hosts do not attach directly to communication wires of the network. Instead each host attaches to one of the packet switches using a serial communication line. In one sense the connection between a host and an X.25 packet switch is a miniature network consisting of one serial link. The host must follow a complicated procedure to transfer packets onto the network.

• *Physical Layer.* X.25 specifies a standard for the physical interconnection between host computers and network packet switches, as well as the procedures used to transfer packets from one machine to another. In the reference model, level *1* specifies the physical interconnection including electrical characteristics of voltage and current. A corresponding protocol, X.21, gives the details used by public data networks.

• *Data Link Layer.* The level *2* portion of the X.25 protocol specifies how data travels between a host and the packet switch to which it connects. X.25 uses the term *frame* to refer to a unit of data as it passes between a host and a packet switch (it is important to understand that the X.25 definition of *frame* differs slightly from the way we have used it). Because raw hardware delivers only a stream of bits, the level *2* protocol must define the format of frames and specify how the two machines recognize frame boundaries. Because transmission errors can destroy data, the level *2* protocol includes error detection (e.g., a frame checksum). Finally, because transmission is unreliable, the level *2* protocol specifies an exchange of acknowledgements that allows the two machines to know when a frame has been transferred successfully.

One commonly used level *2* protocol, named the *High Level Data Link Communication*, is best known by its acronym, *HDLC*. Several versions of HDLC exist, with the most recent known as *HDLC/LAPB*. It is important to remember that successful transfer at level *2* means a frame has been passed to the network packet switch for delivery; it does not guarantee that the packet switch accepted the packet or was able to route it.

†The International Telecommunications Union was formerly named the *Consultative Committee on Inter-*

• *Network Layer.* The ISO reference model specifies that the third level contains functionality that completes the definition of the interaction between host and network. Called the *network* or *communication subnet* layer, this level defines the basic unit of transfer across the network and includes the concepts of destination addressing and routing. Remember that in the X.25 world, communication between host and packet switch is conceptually isolated from the traffic that is being passed. Thus, the network might allow packets defined by level 3 protocols to be larger than the size of frames that can be transferred at level 2. The level 3 software assembles a packet in the form the network expects and uses level *2* to transfer it (possibly in pieces) to the packet switch. Level 3 must also respond to network congestion problems.

• *Transport Layer.* Level 4 provides end-to-end reliability by having the destination host communicate with the source host. The idea here is that even though lower layers of protocols provide reliable checks at each transfer, the end-to-end layer double checks to make sure that no machine in the middle failed.

• *Session Layer.* Higher levels of the ISO model describe how protocol software can be organized to handle all the functionality needed by application programs. The ISO committee considered the problem of remote terminal access so fundamental that they assigned layer 5 to handle it. In fact, the central service offered by early public data networks consisted of terminal to host interconnection. The carrier provides a special purpose host computer called a *Packet Assembler And Disassembler* (*PAD*) on the network with dialup access. Subscribers, often travelers who carry their own computer and modem, dial up the local PAD, make a network connection to the host with which they wish to communicate, and log in. Many carriers choose to make using the network for long distance communication less expensive than direct dialup.

• *Presentation Layer.* ISO layer 6 is intended to include functions that many application programs need when using the network. Typical examples include standard routines that compress text or convert graphics images into bit streams for transmission across a network. For example an ISO standard known as *Abstract Syntax Notation 1* (*ASN.1*), provides a representation of data that application programs use. One of the TCP/IP protocols, SNMP, also uses ASN.1 to represent data.

• *Application Layer.* Finally, ISO layer 7 includes application programs that use the network. Examples include electronic mail or file transfer programs. In particular, the ITU-TS has devised a protocol for electronic mail known as the *X.400* standard. In fact, the ITU and ISO worked jointly on message handling systems; the ISO version is called *MOTIS*.

11.5.1 The TCP/IP Internet Layering Model

The second major layering model did not arise from a standards committee, but came instead from research that led to the TCP/IP protocol suite. With a little work, the ISO model can be stretched to describe the TCP/IP layering scheme, but the underlying assumptions are different enough to warrant distinguishing the two.

Broadly speaking, TCP/IP software is organized into four conceptual layers that build on a fifth layer of hardware. Figure 11.5 shows the conceptual layers as well as the form of data as it passes between them.

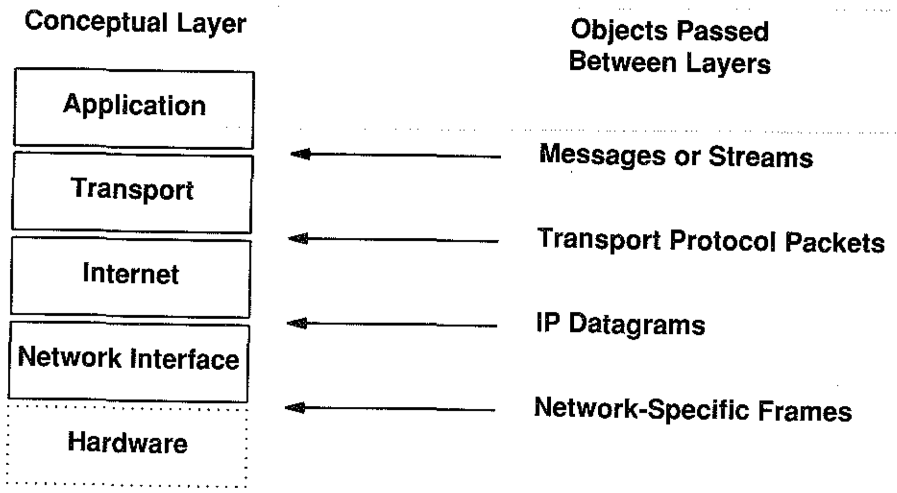


Figure 11.5 The 4 conceptual layers of TCP/IP software and the form of objects passed between layers. The layer labeled *network interface* is sometimes called the *data link* layer.

• *Application Layer*. At the highest level, users invoke application programs that access services available across a TCP/IP internet. An application interacts with one of the transport level protocols to send or receive data. Each application program chooses the style of transport needed, which can be either a sequence of individual messages or a continuous stream of bytes. The application program passes data in the required form to the transport level for delivery.

• *Transport Layer*. The primary duty of the *transport layer* is to provide communication from one application program to another. Such communication is often called *end-to-end*. The transport layer may regulate flow of information. It may also provide reliable transport, ensuring that data arrives without error and in sequence. To do so, transport protocol software arranges to have the receiving side send back acknowledgements and the sending side retransmit lost packets. The transport software divides the stream of data being transmitted into small pieces (sometimes called *packets*) and passes each packet along with a destination address to the next layer for transmission.

Although Figure 11.5 uses a single block to represent the application layer, a general purpose computer can have multiple application programs accessing the internet at one time. The transport layer must accept data from several user programs and send it

to the next lower layer. To do so, it adds additional information to each packet, including codes that identify which application program sent it and which application program should receive it, as well as a checksum. The receiving machine uses the checksum to verify that the packet arrived intact, and uses the destination code to identify the application program to which it should be delivered.

• *Internet Layer*. As we have already seen, the Internet layer handles communication from one machine to another. It accepts a request to send a packet from the transport layer along with an identification of the machine to which the packet should be sent. It encapsulates the packet in an IP datagram, fills in the datagram header, uses the routing algorithm to determine whether to deliver the datagram directly or send it to a router, and passes the datagram to the appropriate network interface for transmission. The Internet layer also handles incoming datagrams, checking their validity, and uses the routing algorithm to decide whether the datagram should be processed locally or forwarded. For datagrams addressed to the local machine, software in the internet layer deletes the datagram header, and chooses from among several transport protocols the one that will handle the packet. Finally, the Internet layer sends ICMP error and control messages as needed and handles all incoming ICMP messages.

• *Network Interface Layer*. The lowest level TCP/IP software comprises a network interface layer, responsible for accepting IP datagrams and transmitting them over a specific network. A network interface may consist of a device driver (e.g., when the network is a local area network to which the machine attaches directly) or a complex subsystem that uses its own data link protocol (e.g., when the network consists of packet switches that communicate with hosts using HDLC).

11.6 Differences Between X.25 And Internet Layering

There are two subtle and important differences between the TCP/IP layering scheme and the X.25 scheme. The first difference revolves around the focus of attention on reliability, while the second involves the location of intelligence in the overall system.

11.6.1 Link-Level vs. End-To-End Reliability

One major difference between the TCP/IP protocols and the X.25 protocols lies in their approaches to providing reliable data transfer services. In the X.25 model, protocol software detects and handles errors at all levels. At the link level, complex protocols guarantee that the transfer between a host and the packet switch to which it connects will be correct. Checksums accompany each piece of data transferred, and the receiver acknowledges each piece of data received. The link level protocol includes timeout and retransmission algorithms that prevent data loss and provide automatic recovery after hardware fails and restarts.

Successive levels of X.25 provide reliability of their own. At level *3*, X.25 also provides error detection and recovery for packets transferred onto the network, using checksums as well as timeout and retransmission techniques. Finally, level *4* must provide end-to-end reliability, having the source correspond with the ultimate destination to verify delivery.

In contrast to such a scheme, TCP/IP bases its protocol layering on the idea that reliability is an end-to-end problem. The architectural philosophy is simple: construct the internet so it can handle the expected load, but allow individual links or machines to lose data or corrupt it without trying to repeatedly recover. In fact, there is little or no reliability in most TCP/IP network interface layer software. Instead, the transport layer handles most error detection and recovery problems.

The resulting freedom from interface layer verification makes TCP/IP software much easier to understand and implement correctly. Intermediate routers can discard datagrams that become corrupted because of transmission errors. They can discard datagrams that cannot be delivered. They can discard datagrams when the arrival rate exceeds machine capacity, and can reroute datagrams through paths with shorter or longer delay without informing the source or destination.

Having unreliable links means that some datagrams do not arrive. Detection and recovery of datagram loss is carried out between the source host and the ultimate destination and is, therefore, called *end-to-end* verification. The end-to-end software located in the transport layer uses checksums, acknowledgements, and timeouts to control transmission. Thus, unlike the connection-oriented X.25 protocol layering, the TCP/IP software focuses most of its reliability control in one layer.

11.6.2 Locus of Intelligence and Decision Making

Another difference between the X.25 model and the TCP/IP model emerges when one considers the locus of authority and control. As a general rule, networks using X.25 adhere to the idea that a network is a utility that provides a transport service. The vendor that offers the service controls network access and monitors traffic to keep records for accounting and billing. The network vendor also handles problems like routing, flow control, and acknowledgements internally, making transfers reliable. This view leaves little that the hosts can (or need to) do. In short, the network is a complex, independent system to which one can attach relatively simple host computers; the hosts themselves participate in the network operation very little.

By contrast, TCP/IP requires hosts to participate in almost all of the network protocols. We have already mentioned that hosts actively implement end-to-end error detection and recovery. They also participate in routing because they must choose a router when sending datagrams, and they participate in network control because they must handle ICMP control messages. Thus, when compared to an X.25 network, a TCP/IP internet can be viewed as a relatively simple packet delivery system to which intelligent hosts attach.

11.7 The Protocol Layering Principle

Independent of the particular layering scheme used, or the functions of the layers, the operation of layered protocols is based on a fundamental idea. The idea, called the *layering principle*, can be summarized succinctly:

> *Layered protocols are designed so that layer* n *at the destination receives exactly the same object sent by layer* n *at the source.*

The layering principle explains why layering is such a powerful idea. It allows the protocol designer to focus attention on one layer at a time, without worrying about how lower layers perform. For example, when building a file transfer application, the designer thinks only of two copies of the application program executing on two machines and concentrates on the messages they need to exchange for file transfer. The designer assumes that the application on one host receives exactly what the application on the other host sends.

Figure 11.6 illustrates how the layering principle works:

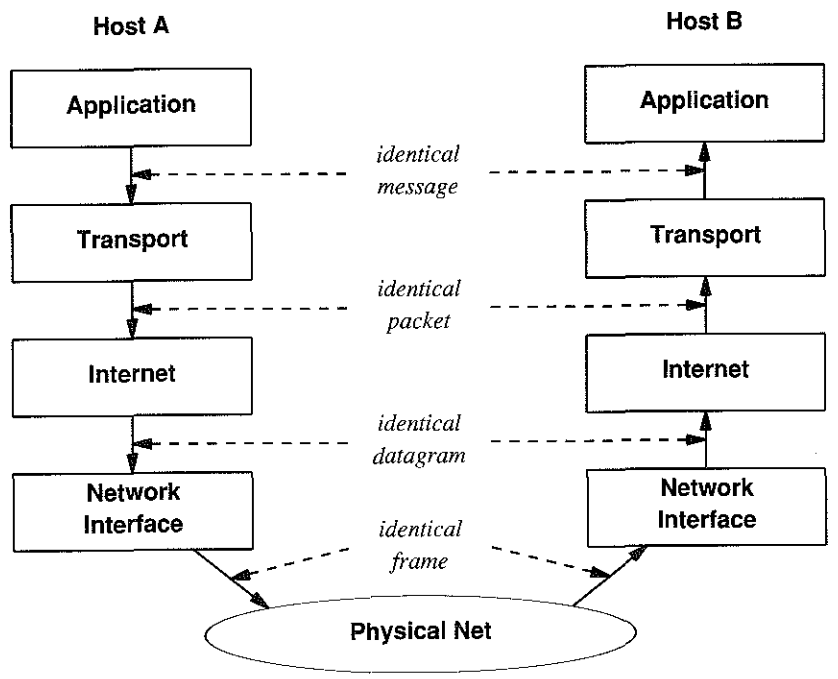


Figure 11.6 The path of a message as it passes from an application on one host to an application on another. Layer *n* on host *B* receives exactly the same object that layer *n* on host *A* sent.

11.7.1 Layering in a TCP/IP Internet Environment

Our statement of the layering principle is somewhat vague, and the illustration in Figure 11.6 skims over an important issue because it fails to distinguish between transfers from source to ultimate destination and transfers across multiple networks. Figure 11.7 illustrates the distinction, showing the path of a message sent from an application program on one host to an application on another through a router.

As the figure shows, message delivery uses two separate network frames, one for the transmission from host *A* to router *R*, and another from router *R* to host *B*. The network layering principle states that the frame delivered to *R* is identical to the frame sent by host *A*. By contrast, the application and transport layers deal with end-to-end issues and are designed so the software at the source communicates with its peer at the ultimate destination. Thus, the layering principle states that the packet received by the transport layer at the ultimate destination is identical to the packet sent by the transport layer at the original source.

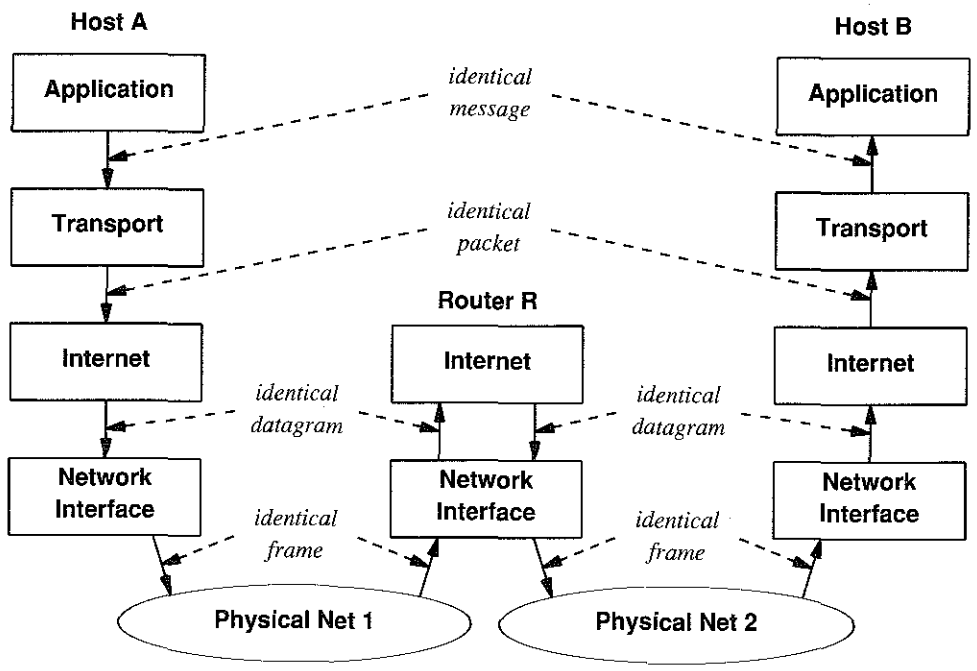


Figure 11.7 The layering principle when a router is used. The frame delivered to router *R* is exactly the frame sent from host *A*, but differs from the frame sent between *R* and *B*.

It is easy to understand that in higher layers, the layering principle applies across end-to-end transfers, and that at the lowest layer it applies to a single machine transfer. It is not as easy to see how the layering principle applies to the Internet layer. On one hand, we have said that hosts attached to an internet should view it as a large, virtual network, with the IP datagram taking the place of a network frame. In this view, datagrams travel from original source to ultimate destination, and the layering principle guarantees that the ultimate destination receives exactly the datagram that the original source sent. On the other hand, we know that the datagram header contains fields, like a *time to live* counter, that change each time the datagram passes through a router. Thus, the ultimate destination will not receive exactly the same datagram as the source sent. We conclude that although most of the datagram stays intact as it passes across an internet, the layering principle only applies to datagrams across single machine transfers. To be accurate, we should not view the Internet layer as providing end-to-end service.

11.8 Layering In The Presence Of Network Substructure

Recall from Chapter 2 that some wide area networks contain multiple packet switches. For example, a WAN can consist of routers that connect to a local network at each site as well as to other routers using leased serial lines. When a router receives a datagram, it either delivers the datagram to its destination on the local network, or transfers the datagram across a serial line to another router. The question arises: "How do the protocols used on serial lines fit into the TCP/IP layering scheme?" The answer depends on how the designer views the serial line interconnections.

From the perspective of IP, the set of point-to-point connections among routers can either function like a set of independent physical networks, or they can function collectively like a single physical network. In the first case, each physical link is treated exactly like any other network in the internet. It is assigned a unique (usually class C) network number, and the two hosts that share the link each have a unique IP address assigned for their connection. Routes are added to the IP routing table as they would be for any other network. A new software module is added at the network interface layer to control the new link hardware, but no substantial changes are made to the layering scheme. The main disadvantage of the independent network approach is that it proliferates network numbers (one for each connection between two machines), causing routing tables to be larger than necessary. Both *Serial Line IP* (*SLIP*) and the *Point to Point Protocol* (*PPP*) treat each serial link as a separate network.

The second approach to accommodating point-to-point connections avoids assigning multiple IP addresses to the physical wires. Instead, it treats all the connections collectively as a single, independent IP network with its own frame format, hardware addressing scheme, and data link protocols. Routers that use the second approach need only one IP network number for all point-to-point connections.

Using the single network approach means extending the protocol layering scheme to add a new intranetwork routing layer between the network interface layer and the hardware devices. For machines with only one point-to-point connection, an additional layer seems unnecessary. To see why it is needed, consider a machine with several physical point-to-point connections, and recall from Figure 11.2 how the network interface layer is divided into multiple software modules that each control one network. We need to add one new network interface for the new point-to-point network, but the new interface must control multiple hardware devices. Furthermore, given a datagram to send, the new interface must choose the correct link over which the datagram should be sent. Figure 11.8 shows the organization.

The Internet layer software passes to the network interface all datagrams that should be sent out on any of the point-to-point connections. The interface passes them to the intranet routing module that must further distinguish among multiple physical connections and route the datagram across the correct one.

The programmer who designs the intranet routing software determines exactly how the software chooses a physical link. Usually, the algorithm relies on an intranet routing table. The intranet routing table is analogous to the internet routing table in that it specifies a mapping of destination address to route. The table contains pairs of entries, (*D*, *L*), where *D* is a destination host address and *L* specifies one of the physical lines used to reach that destination.

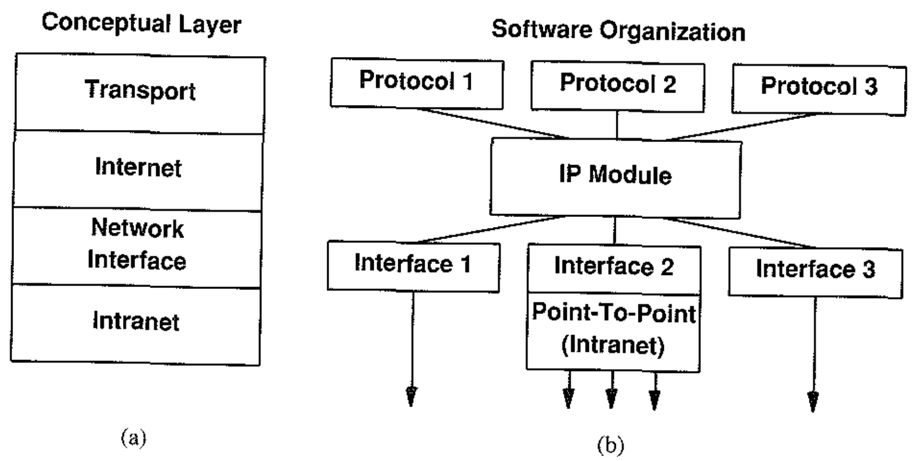


Figure 11.8 (a) conceptual position of an intranet protocol for point-to-point connections when IP treats them as a single IP network, and (b) detailed diagram of corresponding software modules. Each arrow corresponds to one physical device.

The difference between an internet routing table and an intranet routing table is that intranet routing tables are quite small. They only contain routing information for hosts directly attached to the point-to-point network. The reason is simple: the Internet layer maps an arbitrary destination address to a specific router address before passing the datagram to a network interface. Thus, the intranet layer is asked only to distinguish among machines on a single point-to-point network.

11.9 Two Important Boundaries In The TCP/IP Model

The conceptual protocol layering includes two boundaries that may not be obvious: a protocol address boundary that separates high-level and low-level addressing, and an operating system boundary that separates the system from application programs.

11.9.1 High-Level Protocol Address Boundary

Now that we have seen the layering of TCP/IP software, we can be precise about an idea introduced in Chapter 8: a conceptual boundary partitions software that uses low-level (physical) addresses from software that uses high-level (IP) addresses. As Figure 11.9 shows, the boundary occurs between the network interface layer and the Internet layer. That is,

Application programs as well as all protocol software from the Internet layer upward use only IP addresses; the network interface layer handles physical addresses.

Thus, protocols like ARP belong in the network interface layer. They are not part of IP.

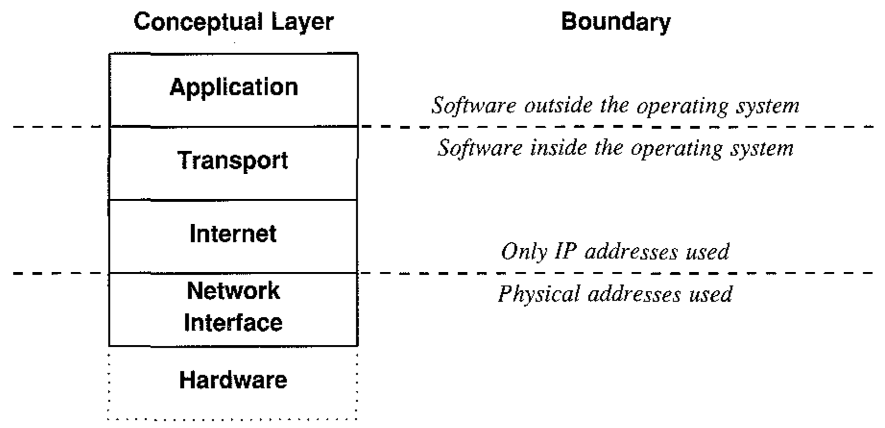


Figure 11.9 The relationship between conceptual layering and the boundaries for operating system and high-level protocol addresses.

11.9.2 Operating System Boundary

Figure 11.9 shows another important boundary as well, the division between software that is generally considered part of the operating system and software that is not. While each implementation of TCP/IP chooses how to make the distinction, many follow the scheme shown. Because they lie inside the operating system, passing data between lower layers of protocol software is much less expensive than passing it between an application program and a transport layer. Chapter 20 discusses the problem in more detail and describes an example of the interface an operating system might provide.

11.10 The Disadvantage Of Layering

We have said that layering is a fundamental idea that provides the basis for protocol design. It allows the designer to divide a complicated problem into subproblems and solve each one independently. Unfortunately, the software that results from strict layering can be extremely inefficient. As an example, consider the job of the transport layer. It must accept a stream of bytes from an application program, divide the stream into packets, and send each packet across the internet. To optimize transfer, the transport layer should choose the largest possible packet size that will allow one packet to travel in one network frame. In particular, if the destination machine attaches directly to one of the same networks as the source, only one physical net will be involved in the transfer, so the sender can optimize packet size for that network. If the software preserves strict layering, however, the transport layer cannot know how the Internet module will route traffic or which networks attach directly. Furthermore, the transport layer will not understand the datagram or frame formats nor will it be able to determine how many octets of header will be added to a packet. Thus, strict layering will prevent the transport layer from optimizing transfers.

Usually, implementors relax the strict layering scheme when building protocol software. They allow information like route selection and network MTU to propagate upward. When allocating buffers, they often leave space for headers that will be added by lower layer protocols and may retain headers on incoming frames when passing them to higher layer protocols. Such optimizations can make dramatic improvements in efficiency while retaining the basic layered structure.

11.11 The Basic Idea Behind Multiplexing And Demultiplexing

Communication protocols uses techniques of *multiplexing* and *demultiplexing* throughout the layered hierarchy. When sending a message, the source computer includes extra bits that encode the message type, originating program, and protocols used.

Eventually, all messages are placed into network frames for transfer and combined into a stream of packets. At the receiving end, the destination machine uses the extra information to guide processing.

Consider an example of demultiplexing shown in Figure 11.10.

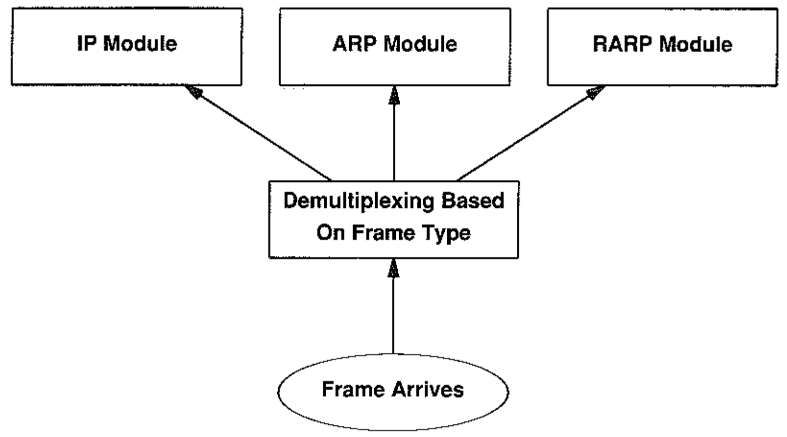


Figure 11.10 Demultiplexing of incoming frames based on the type field found in the frame header.

The figure illustrates how software in the network interface layer uses the frame type to choose a procedure that handles the incoming frame. We say that the network interface *demultiplexes* the frame based on its type. To make such a choice possible, software in the source machine must set the frame type field before transmission. Thus, each software module that sends frames uses the type field to specify frame contents.

Multiplexing and demultiplexing occur at almost every protocol layer. For example, after the network interface demultiplexes frames and passes those frames that contain IP datagrams to the IP module, the IP software extracts the datagram and demultiplexes further based on the transport protocol. Figure 11.11 demonstrates demultiplexing at the Internet layer.

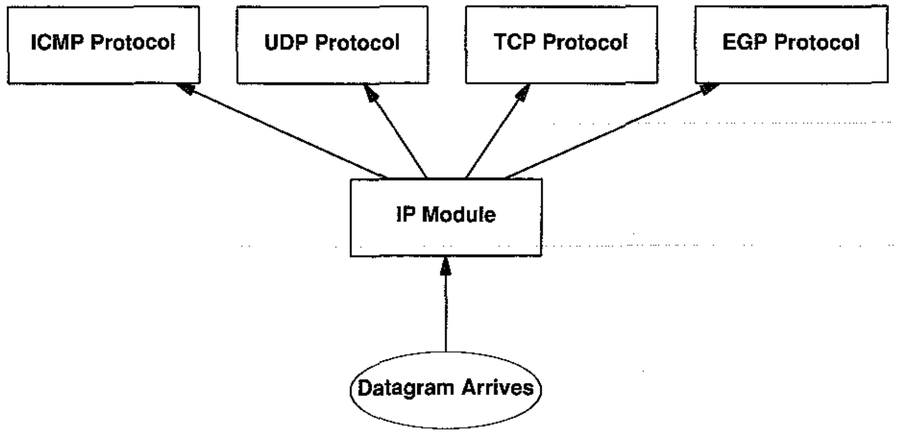


Figure 11.11 Demultiplexing at the Internet layer. IP software chooses an appropriate procedure to handle a datagram based on the protocol type field in the datagram header.

To decide how to handle a datagram, internet software examines the header of a datagram and selects a protocol handler based on the datagram type. In the example, the possible datagram types are: *ICMP*, which we have already examined, and *UDP*, *TCP*, and *EGP*, which we will examine in later chapters.

11.12 Summary

Protocols are the standards that specify how data is represented when being transferred from one machine to another. Protocols specify how the transfer occurs, how errors are detected, and how acknowledgements are passed. To simplify protocol design and implementation, communication problems are segregated into subproblems that can be solved independently. Each subproblem is assigned a separate protocol.

The idea of layering is fundamental because it provides a conceptual framework for protocol design. In a layered model, each layer handles one part of the communication problem and usually corresponds to one protocol. Protocols follow the layering principle, which states that the software implementing layer *n* on the destination machine receives exactly what the software implementing layer *n* on the source machine sends.

We examined the 4-layer Internet reference model as well as the ISO 7-layer reference model. In both cases, the layering model provides only a conceptual framework for protocol software. The ITU-TS X.25 protocols follow the ISO reference model and provide an example of reliable communication service offered by a commercial utility, while the TCP/IP protocols provide an example of a different layering scheme.

In practice, protocol software uses multiplexing and demultiplexing to distinguish among multiple protocols within a given layer, making protocol software more complex than the layering model suggests.

FOR FURTHER STUDY

Postel [RFC 791] provides a sketch of the Internet Protocol layering scheme, and Clark [RFC 817] discusses the effect of layering on implementations. Saltzer, Reed, and Clark [1984] argues that end-to-end verification is important. Chesson [1987] makes the controversial argument that layering produces intolerably bad network throughput. Volume 2 of this text examines layering in detail, and shows an example implementation that achieves efficiency by compromising strict layering and passing pointers between layers.

The ISO protocol documents [1987a] and [1987b] describe ASN.1 in detail. Sun [RFC 1014] describes XDR, an example of what might be called a TCP/IP presentation protocol. Clark discusses passing information upward through layers [Clark 1985].

EXERCISES

11.1 Study the ISO layering model in more detail. How well does the model describe communication on a local area network like an Ethernet?

11.2 Build a case that TCP/IP is moving toward a five-level protocol architecture that includes a presentation layer. (Hint: various programs use the XDR protocol, Courier, and ASN.1.)

11.3 Do you think any single presentation protocol will eventually emerge that replaces all others? Why or why not?

11.4 Compare and contrast the tagged data format used by the ASN.1 presentation scheme with the untagged format used by XDR. Characterize situations in which one is better than the other.

11.5 Find out how UNIX systems uses the *mbuf* structure to make layered protocol software efficient.

11.6 Read about the System V UNIX *streams* mechanism. How does it help make protocol implementation easier? What is its chief disadvantage?